(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 762 273 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
12.03.1997 Bulletin 1997/11

(51) Int. Cl.[6]: **G06F 9/44**, G06F 9/46

(21) Application number: **96114176.9**

(22) Date of filing: **04.09.1996**

(84) Designated Contracting States:
DE FR GB IT

(30) Priority: **06.09.1995 JP 229545/95**
12.10.1995 JP 264193/95

(71) Applicant: **SEIKO EPSON CORPORATION**
Tokyo 163 (JP)

(72) Inventors:
- **Kimura, Yoshihiro**
  Suwa-shi, Nagano-ken, 392 (JP)
- **Hisamatsu, Yutaka**
  Suwa-shi, Nagano-ken, 392 (JP)

(74) Representative: **Hoffmann, Eckart, Dipl.-Ing.**
Patentanwalt,
Bahnhofstrasse 103
82166 Gräfelfing (DE)

(54) Peripheral device control system using a plurality of objects, and programming method therefor

(57) A control system built around plural common objects is provided with greater flexibility and easy customizability, and a programming method for said control system is provided.
An interface object (15) capable of two-way communications is created and used when first OCX (10), a common object, creates and controls second OCX (11). This interface object (15) is able to return events generated by second OCX (11) to first OCX (10), and first OCX (10) is able to completely control the operation of second OCX (11).

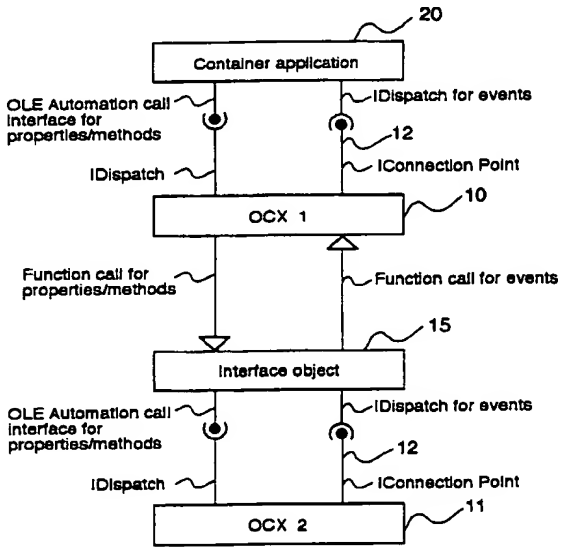

FIG. 1

EP 0 762 273 A1

Description

The present invention relates to a control system comprising plural common objects that can be shared by plural application programs, and relates specifically to a control system that can be customized using said common objects, to a programming method for said control system, and to a peripheral devices control system applying said control system and programming method.

Computers are used today to execute a wide range of processes and use a variety of peripheral equipment that can be connected to computers. It is therefore necessary to build and provide flexible computerized systems that can be customized to the user's working environment and needs. For example, it is possible today to connect different types of printers, displays, bar code readers, and other input/output devices using a common standard bus. When the software needed to control these I/O devices is installed in the computer, the control system must be able to appropriately control each of these many devices. It is also easy to build a networked computing environment in which plural computers are connected to a common network. This makes it desirable to be able to control the processes and procedures executed by each computer on the network.

One method of constructing this type of flexible system executed in a computer is to create plural common objects that can be shared, and have the operating system and application programs use these common objects. An example of such a method is the Object Linking and Embedding (OLE) technology developed by Microsoft Corporation, and the related OLE Automation or OLE custom controls (OCX) which are interfaces for OLE programming.

One problem with such programming methods, however, is that when one common object uses another common object, the functions of both common objects cannot be fully utilized. Using the custom controls OCX described above, for example, the client OCX can access properties and methods from the server OCX, but the client OCX cannot receive events generated by the server OCX.

This means that if the first or client OCX is an object for controlling the functions of a peripheral device such as a printer, and the second or server OCX is a driver for the type of peripheral device controlled by the first OCX, the first OCX can control the functions of the peripheral device via the second OCX but it cannot receive peripheral device status information through the second OCX.

The object of the present invention is therefore to resolve this problem by providing a method and system for easily constructing a variety of operating systems and application programs using plural common objects that can be shared by plural application programs. More specifically, the object of the present invention is to provide a method and system that improve the interface between common objects and to enable each common object to fully utilize the functions of other common objects.

For example, when one common object is the client or controller and another common object is the server, events generated by the server object can be recognized by the client or controller object, and to provide a method for constructing such systems.

The present invention specifically applies to a system comprising plural control objects used as common objects, each comprising a first function for accessing properties including attribute values and for invoking methods to call implemented functions and a second function for posting events including asynchronously occurring actions.

When a control system executable by a computer is constructed by means of this system, at least one second control object is created by a first control object with the first control object controlling the second object. In this type of control system, the present invention provides an interface object having a function for communicating properties or methods between the first control object and the second object, and a function for communicating events from the second control object to the first control object.

It is possible by means of the present invention for a first control object and a second common object to communicate all properties, methods, and events therebetween by means of a two-way communications interface object. It is therefore simple to construct a control system capable of fully utilizing functions made available as control objects.

In a control system of the present invention according to a first embodiment, the first control object can receive events generated by a second control object, execute processes appropriate to the received event, and rapidly communicate said event to another control object, application program, or operating system using said first control object. The present invention is therefore able to build a control system that fully utilizes the functions of all common objects.

A control system according to an alternative embodiment of the present invention provides a second interface for receiving events to a control object comprising the functionality of a common object. The control object comprises a function for informing an application program or other control object of specific properties, including attribute values, or methods for calling implemented functions, and comprises a first interface for posting events including asynchronous actions.

In a control system according to the alternative embodiment of the present invention, a first control object comprises a second interface for receiving events. Events occurring asynchronously in a second control object can be immediately communicated through the first interface and second interface to the first control object, or to higher level application programs. It is therefore possible to easily construct a variety of control systems by means of a first control object creating a second control object, and appropriately controlling the first control object and the second control

object, thereby achieving a high speed, multiple function control system.

The present invention will become more fully understood from the detailed description of preferred embodiments given below and the accompanying diagrams wherein:

Fig. 1 is a block diagram of the basic configuration of a control system according to the first embodiment of the present invention.

Fig. 2 is a flow chart showing the process for confirming a connection between an interface object and a control object in the control system shown in Figs. 1 and 7.

Fig. 3 is a flow chart showing the process for establishing a connection and event correspondence between an interface object and a control object in the control system shown in Figs. 1 and 7.

Fig. 4 is a flow chart showing the process whereby a control object notifies the interface object of an event in the control system shown in Figs. 1 and 7.

Fig. 5 is a block diagram of a control system using a variation of the first embodiment of the present invention.

Fig. 6 is a block diagram of the system for controlling the peripheral devices of a point-of-sale (POS) system constructed with a control system using the first embodiment of the present invention.

Fig. 7 is a block diagram of the basic configuration of a control system according to the second embodiment of the present invention.

Fig. 8 is a block diagram of the system for controlling the peripheral devices of a point-of-sale (POS) system constructed with a control system using the second embodiment of the present invention.

Embodiment 1

The first embodiment of the invention is described in detail below based on an implementation for the Microsoft Foundation Class (MFC), an applications development environment published by Microsoft.

MFC provides various libraries used to facilitate OLE programming, and makes it possible to develop systems using OLE Automation functions written in object-oriented programming languages such as Microsoft's Visual C++. As shown in Fig. 1, these systems are constructed with hierarchical links between the application program (container application) 20 and common objects OCX 10 and 11 written to enable shared use throughout the system.

These common objects OCX 10 and 11 can be provided as components of executable software (EXE) modules, or as Dynamic Link Library (DLL) modules. Executable software and DLL modules providing these common object services are known as EXE servers or DLL servers; the container application 20 is known as a client or controller. Common objects used in the OLE architecture may be derived from the CCmdTarget class provided in the MFC library, for example. The CCmdTarget class supports the IUnknown interface for managing common objects and the interfaces needed as common objects. The CCmdTarget class also supports the IDispatch interface which delivers structures such as data storage structures from the controller, and which can access common object properties and methods by calling the member function Invoke.

Note that properties are attributes of the common object. Properties of an object associated with a printer, for example, may include ink colors, text, numbers, fonts, and the operation executed when a given push-button or key is pressed. Methods are functions, such as editing functions, implemented in the common object and are called to access and use functions of the common object.

In addition to providing access to properties and methods for client objects, common objects also announce events. Events are asynchronous external actions, such as clicking a mouse button or pressing a key, accessing the common object. A common object normally announces an event by calling the Invoke function of the controller in a manner similar to the way in which the controller accesses properties and invokes methods. If the connection point for the event does not have a valid interface, the event will be ignored. Referring to Fig. 1, IConnectionPoint 12 is the interface that provides a connection for announcing events.

Container application 20 in Fig. 1 is a control system using a common object. In the control system shown in this figure, container application 20 creates a first object OCX 10, and this first OCX 10 creates a second object OCX 11 for control. A COleDispatchDriver class is included in the MFC library to handle the complexities of calling the Invoke function from a container application or OCX. By calling member functions of objects derived from this class, first OCX 10 can create the second OCX 11 which is another control object, and can then use the properties and methods of the second OCX 11.

When common objects in the server are used, it is possible, for example, that a given common object may reside on a processor different from the processor on which the container application is executing. In such cases, the control system of the present embodiment can be constructed by executing a communications process on each processor where the controller and the server reside. It is also possible to copy the common objects used on one processor to build a control system. In either case, however, it is not possible to modify the code of common objects in any way, and memory is reserved each time the common object is activated. The control system communicates between the server-side common objects and the controller using the reserved memory.

It is therefore possible for the controller to use the same common object multiple times, in which case the common object need not be copied multiple times, but the data, stack, and other requisite memory areas are reserved for each use. In this specification, each distinct use of a common object, regardless how it is implemented, is referred to as an object.

When the first OCX 10 is the controller and it calls a second OCX 11, it can use an interface object derived from the COleDispatchDriver class as described above. Note that the term "interface object" refers to distinct uses of interface objects, regardless how they are created, similar to that described above for common objects. The Visual C++ 2.0 COleDispatchDriver class supports a function for passing methods and properties to the second OCX through the IDispatch interface, but it does not support a function for capturing events returned by the second OCX. This does not create a problem when the container application 20 recognizes the second OCX 11 used by the first OCX 10 and comprises a function for capturing events of the second OCX 11. When the container application 20 does not recognize the second OCX 11 used by the first OCX 10, however, it is not possible to capture the events generated by the second OCX 11.

When a common object is already identified, or a common object is prepared for a specific container application, it is possible either to construct the control system such that a common object can pass events to the container application, or to control processing such that the second OCX 11 does not generate events affecting the container application. It is desirable, however, to have the ability to construct a more flexible control system to support different external input/output devices and adapt to a wide range of computing environments. It is therefore necessary to support an environment in which one common object can be customized using another common object, e.g., an environment in which the container application 20 can be customized using common objects.

Referring again to Fig. 1, the present invention provides an interface object 15 enabling two-way communications between one common object and another common object, thereby enabling the construction of more flexible control systems. If common objects are linked using an interface object enabling two-way communications, there is no need for container application 20 to recognize the second OCX 11, and the second OCX 11 can be provided as an object unaffected by the container application 20.

To achieve an interface object providing complete support even for events, the present embodiment provides a COcxDispatchDriver class handling dispatch processing for one OCX to use another OCX. This COcxDispatchDriver class is an object class derived with multiple inheritance from the CCmdTarget class and the COleDispatchDriver class, and may be represented by a C++ statement as follows:

```
class COcxDispatchDriver: public CCmdTarget, public COleDispatchDriver {...};
```

Various classes and declarations in the examples shown herein conform to definitions provided in the MFC library mentioned above and in a Windows Software Developers Kit (SDK) provided by Microsoft. Other implementations are possible.

As described above, the CCmdTarget class is the class for deriving objects with a server function and it supports the IDispatch interface. It is therefore possible to achieve an interface object that operates as a server to second OCX 11 while supporting an interface through which events can be passed.

The COleDispatchDriver class is an object class that operates as either client or controller for the second OCX 11 and it comprises a function for easily accessing methods and properties.

The COcxDispatchDriver classis an object class comprising the two main functions described above while inheriting and supporting all of the functions of both parent classes. The COcxDispatchDriver class is appropriate as an object class generating an interface object according to the present invention. Because there is no overlap between the operation and function of the CCmdTarget class and the COleDispatchDriver class, there is also no problem created by multiple inheritance.

The specification for the derived COcxDispatchDriver class is as follows;

```
Class COcxDispatchDriver: public CCmdTarget, public COleDispatchDriver {
COcxDispatchDriver::COcxDispatchDriver(); // create
public:              // member data
UINT m_ObjID;        //object ID initialized uniquely by EstablishConnection()
IID m_IIDEvents;     // event interface ID
UINT m_nEvents;      // number of events
CDWordArray m_dispID  // array for converting dispatch ID using user-defined dispatch
                     map
public:              // member functions
BOOL EstablishConnection(REFCLSID clsid, COleException *pError=NULL);
                     // creates an object, and enables an access connection to methods,
                     properties, and events
void DestroyConnection(); // releases the connection, and deletes the object.
}
```

To achieve a general-purpose class, the COcxDispatchDriver class does not include an event processing function (event handler). It is therefore necessary to describe an event handler for specific OCX events in any objects derived from COcxDispatchDriver. This is accomplished as follows.

A dispatch map is first described according to the external name of the event(s) returned by the OCX. The sequence of events in the dispatch map may be freely ordered, and the correlation to the sequence of events actually received is established when the event connection is established. Note that it is necessary to fully describe all OCX events.

Dispatch map entries may be described using a macro call such as follows with the major parameters being, in sequence, the dispatch driver class name, external event name, event handler name, return values, and parameter data:

DISP_FUNCTION(_DSoprn, "ControlCompleteEvent", ControlCompleteEvent, VT_EMPTY, VTS_14 VTS_SCODE VTS_PBSTR VTS_I4)

By describing the dispatch map according to the external event name, events created by the OCX can be received in an intuitive, easily understood format. Program development can be greatly facilitated by declaring the event handler using a normal function prototype statement as follows:

void _DSoprn::ControlCompleteEvent(long ControlID, SCODE Result, BSTR FAR*, pString, long data);

The process for setting the connection between second OCX 11 and interface object 15 providing an interface derived from the COcxDispatchDriver class is described below with reference to the flow charts in Fig. 2 and Fig. 3. By simply calling the member function COcxDispatchDriver::EstablishConnection(), the interface object 15 of the present embodiment creates an object (second OCX), supplies the methods and properties, and establishes a connection for receiving events.

When EstablishConnection() is called, the second OCX 11 defined by clsid (the variable storing a 128-bit value identifying the OCX) is created and begins executing as shown in step 31 of Fig. 2. When the operation of second OCX 11 is confirmed in step 32, the IDispatch interface of this OCX 11 is obtained and stored in step 33. The interface object 15 can call the Invoke function and serve methods and properties to the OCX 11, through the IDispatch interface. If the establishment of a connection for passing properties and events to the second OCX 11 is confirmed in step 34, the event interface of the second OCX 11 is checked in step 35.

If an event interface is confirmed in step 36, the event-related data is stored in step 37. The event name list prepared in second OCX 11 is stored to EntryNames, the dispatch ID list is stored to DispIDs, and the parameter data list is stored to ParamInfo. If these processes are confirmed to have executed and completed normally in step 38, the process defining the correlation between the dispatch map and events is executed according to the flow chart shown in Fig. 3. If, however, a connection cannot be established, an error processing task displaying an error message, for example, is executed in step 39.

The process defining the correlation between the dispatch map and events is described next using the flow chart shown in Fig. 3.

This process starts by clearing the counter fcnt to zero in step 41. The counter fcnt is then read in step 42. Step 43 checks whether the event name EntryNames(fcnt) obtained from second OCX 11 is in the dispatch map pDispMap pre-

viously defined in the event handler of interface object 15. If the event name EntryNames(fcnt) is in the dispatch map pDispMap, it is then confirmed in step 44 whether the parameter data ParamInfo(fcnt) matches the dispatch map pDispMap previously defined in the event handler.

If step 44 returns YES, step 45 then defines array m_dispID is defined for converting the dispatch ID DispIDs(fcnt) of the event obtained from second OCX 11 to an index in the dispatch map pDispMap previously defined in the event handler. The counter fcnt is then incremented in step 46, and the loop from step 42 to step 46 is repeated for each remaining event.

When the correlation to the dispatch map pDispMap is established, step 47 sets a connection for events from second OCX 11. In this step the event connection is set by passing the address of the IDispatch reserved in interface object 15 for receiving events to IConnectionPoint, which is the interface for object-to-object connections. This makes it possible to pass events to interface object 15 by calling the Invoke function of the second OCX 11. More specifically, it is possible for interface object 15 to receive events passed from second OCX 11.

Step 48 checks whether the connection is successfully established, the array m_dispID for converting the event dispatch ID is stored in step 49.

If the preceding steps are all completed normally and the connection between interface object 15 and second OCX 11 is set, it is possible from step 50 to pass the methods and properties of second OCX 11 to first OCX 10, and to pass events from second OCX 11 to first OCX 10. In practice interface object 15 must notify first OCX 10 that an event has been received from second OCX 11. A public member function that is called when interface object 15 receives an event is therefore provided in first OCX 10, and interface object 15 calls this member function accordingly.

It is therefore possible by using interface object 15 of the present embodiment for two-way communications between common objects to be maintained. This makes it possible to construct a more flexible control system using common objects.

Note that the interface object of the present embodiment as just described establishes a cross-reference between the event array of the common object and the event array of the interface object, thereby establishing a correlation between the arrays when establishing the event connection. This correlation makes it possible to set a connection if the types and numbers of events match even if first OCX 10 is not fully informed about the second OCX 11 interface. In other words, two-way connections between plural common objects can be established if the properties, methods, and event names match. Developing common objects is therefore easier, and flexible systems offering greater security and reliability can be constructed even when updated components and different objects are used.

Interface object development is also significantly easier and requires less time because the event handler that is activated when an event is received can be described using normal function syntax.

Fig. 4 is a flow chart of the process executed when Invoke(EventID,...) is called from second OCX 11 to post events of the IDispatch interface of interface object 15. The IDispatch interface provides Invoke() and the basic functions supporting Invoke() (including AddRef(), Release(), and QueryInterface(IID&, LPUNKNOWN*)), and responds through QueryInterface() to the interface ID (IID) of the event generated by the other OCX.

When Invoke() is called, the structure storing the event parameters and the dispatch ID defining the current event type are passed to interface object 15. Interface object 15 then checks in step 60 whether the parameters are valid. If the parameters are valid, step 61 converts the dispIDMember array of events obtained from second OCX 11 according to the prepared event handler dispatch map pDispMap based on the conversion array m_dispID. The event handler corresponding to dispIDMember is then extracted from pDispMap in step 62. Step 63 confirms whether the event handler is valid or not.If the event handler is valid, the event handler corresponding to the event is called in step 64 and the process defined by the event handler, e.g., a process informing first OCX 10 of the event, is executed.

A variation of the first embodiment is shown in Fig. 5. In this variant control system, first OCX 10 generates plural objects 11a and 11b of the second OCX with first OCX 10 controlling both second OCX objects 11a and 11b. More specifically, first OCX 10 is the client or controller for the two second OCX objects 11a and 11b. In this control system the interface between first OCX 10 and second OCX objects 11a and 11b is provided by means of the interface objects 15a and 15b that created the two object objects 11a and 11b.

Each time an interface object derived from the COcxDispatchDriver class is generated, a unique ID is generated identifying each interface object. When an event is relayed from the interface object to first OCX 10, the first OCX 10 is able to reference the ID unique to each interface object. This ID is a unique value assigned when the interface object is created, but may as necessary be changed to an appropriate value by the interface object creator, i.e., by the first OCX in the present embodiment.

When plural objects 11a and 11b of the second OCX are created as shown in Fig. 5, events generated separately by second OCX objects 11a and 11b are transmitted to the first OCX via interface objects 15a and 15b. The first OCX can therefore clearly determine which of the two second OCX objects generated the events by referencing the unique ID assigned to each interface object. It is therefore possible as described above using the interface object of the present embodiment for a common object (first OCX) to spawn and use multiple second objects (second OCX). Even in this case, the creator (first OCX) of the plural objects can identify which second OCX object generated an event without individually managing the plural objects.

The event handlers are not contained in the first OCX object that spawned the plural second OCX objects, but rather are contained in each of the interface objects. This makes it possible for the creator (first OCX in this embodiment) to reliably determine what events were generated with what parameters by which one of the second OCX objects.

It is therefore sufficient for the second OCX objects to simply post the respectively generated events to the corresponding interface objects, and it is not necessary for each object to determine whether that object is the first object or whether other objects of the same class exist.

By using an interface object according to the present invention, a control system comprising plural common objects can be flexibly and easily constructed. It is only necessary to provide each common object with minimal information about the other common objects it will use, and each common object needs only minimal information about the other common objects used. It follows that even greater encapsulation of individual common objects is possible. At the same time, the interface object according to the present invention comprises a two-way communications function capable of communicating events, and can thereby fully manipulate other common objects while obtaining full benefit of the functions provided by the other common objects.

Although a two-layer control system comprising first and second custom controls OCX has been described in the preceding embodiments of the invention, control systems with a deeper hierarchical structure of three or more layers can also be constructed in the same manner.

It is also possible to achieve an interface between an application program and a first custom control OCX using the interface object described above. It will also be obvious that such control systems shall not be limited to application programs and may include operating systems.

Furthermore, while the preceding embodiments have been described as achieving the object interface of the present invention under the MFC environment supplied by Microsoft, an object interface comprising the same functionality can be created without using the MFC library. Moreover, systems using common objects substantially equivalent to those described above can be constructed under any system environment using shared common objects; therefore, the invention is not limited to implementations developed by Microsoft OLE Automation or any other specific developement tools.

Systems using common objects as described can be achieved on stand-alone computers using a single processor, on a system comprising multiple processors, or on multiple systems connected in a network environment.

By using the interface object described, the present embodiment provides a control system offering great extensibility and easy program development. It will be obvious, however, that a common object comprising both the described interface object and the functions of the first custom control OCX can be achieved, and a control system enabling two-way communication of properties, methods, and events can be constructed with a system using this OCX.

Application in a peripheral device control system

An example of a peripheral device control system comprising an interface object according to the first embodiment above and plural common objects is shown in Fig. 6. The specific control system shown in Fig. 6 is a point-of-sale (POS) system built around a personal computer (PC) 70. The POS application program 71 is installed in PC 70 and operates under operating system (OS) 105. OS 105 has a function for controlling the peripheral devices normally required by a personal computer using a keyboard driver 106, display monitor driver 107, and other device drivers. Data transfers between POS application program 71, the keyboard, monitor, and other devices are controlled by OS 105.

In addition to these peripheral devices normally provided with any personal computer, a POS system also typically comprises a customer display 110 for displaying the purchase amount and other information for customer confirmation; a receipt printer 112 for printing purchase receipts, for example; a slip printer 113 for imprinting checks and other slips; and a cash drawer 115 for holding money. These peripheral devices are connected to an RS-232C port or other input/output port. For example, customer display 110 is connected to the RS-232C port and printer 111 comprising both receipt printer 112 and slip printer 113 is connected through the customer display 110. Cash drawer 115 is placed below printer 111 and is operated through the control circuitry of printer 111.

Numerous manufacturers market a variety of such peripheral devices, enabling the user to select the devices best suited to the user's POS system architecture and application. Because the specifications of different peripheral device makers and models differ widely, however, it is extremely difficult to construct an application program suited to all available peripheral devices. Specifications may also change as peripheral devices are upgraded. It is therefore conventionally difficult for users to construct POS systems using peripheral devices selected at the convenience of the user. When individual peripheral devices are upgraded, the new version of a currently-used peripheral device is also not necessarily immediately adaptable to existing POS systems.

When the control system is constructed using the custom controls OCX and interface object of the present invention as described above, an extremely open system can be constructed. It is therefore possible to simply construct a POS system using various different models of peripheral devices. It is also simple to accommodate updated versions of the peripheral devices. For example, peripheral device control system 72 of the present embodiment shown in Fig. 6 comprises three levels of custom controls OCX. The first OCX level has the receipt printer format conversion OCX 73

and a slip printer format conversion OCX 74.

OCX 73 and 74 execute the process arranging the data sent from the POS application program 71, e.g., the list of items sold and the total price, to a specific printing format. The specific printing format may be defined by OCX 73 or 74, or the formatting properties may be defined by lower-level custom controls OCX, i.e., receipt printer control OCX 75 or slip printer control OCX 76 in this example, with conversion OCX 73 or 74 doing the actual formatting using the properties of these custom controls OCX. In either case, POS application program 71 only needs to pass the output data to conversion OCX 73 or 74 irrespective of the print out format. The interface format can therefore be limited to provide an application program with high general utility.

It is also possible to convert data output from the application program in a specific format by a first level OCX to a format common to the lower OCX. This makes it possible to increase the general utility of application programs separately developed using the custom control objects OCX.

Level 2 custom controls OCX include receipt printer control OCX 75, slip printer control OCX 76, cash drawer control OCX 77, and customer display control OCX 78. These custom control objects OCX 75 - 78 provide a predetermined application programming interface (API) to the application program and other high-level OCX. The application program and high-level OCX can therefore simply supply data according to a predetermined API specification irrespective of the manufacturers and models of the printers and other peripheral devices that are part of the POS system. The custom controls OCX 75 - 78 on this level convert data input according to a common API specification to the data format of the lower-level OCX, i.e., to the specifications of the peripheral devices composing the actual system, using the properties of the low driver-level OCX reflecting the specifications of the individual peripheral devices.

Level 3 custom controls OCX include printer driver OCX 91 and 92, cash drawer driver OCX 93, and display driver OCX 94. These custom controls OCX 91 - 94 are common objects supplied with and corresponding to each of the peripheral devices, each common object normally differing according to the manufacturer or model of the peripheral device. These driver-level OCX 91 - 94 describe, for example, the maximum number of printing lines, the line pitch, and other printer-specific characteristics and settings (i.e., the printer status) as device properties. These properties can then be referenced by higher-level objects or the application program. The driver-level objects also define methods for outputting print commands, including commands to print the text string at a specific position. These driver-level custom controls OCX output commands specific to the corresponding peripheral device through the appropriate port driver. The printer OCX, for example, outputs the line feed distance (i.e., line pitch), line feed commands, print data and print commands, and automatic paper cutter commands in a predetermined sequence to the printer through port driver 100 based on the specified print position and printing string.

Level 3 OCX also receive asynchronously occurring events (actions) back from the corresponding peripheral device. The printer driver, for example, receives the process result and error status sent from the printer. The contents of these actions are then interpreted based on the specifications of the corresponding peripheral device, converted to a universal format, and returned to the higher-level OCX as an event.

High and low level OCX are linked by means of interface objects 81a, 81b, 82a - 82d, each of which is capable of two-way communications, in the control system of the present embodiment. In addition to methods and properties, events can also be communicated. Level 2 control OCX 75 - 78 receive these events and pass the events, either directly or after conversion to a universal format, through interface objects 82a - 82d to a higher level OCX or application program. Based on the received event, the application program or high level OCX can then output a feedback message to the user, initiate an error processing routine, or execute another defined process. It is therefore possible in the control system according to the present embodiment to quickly execute the process appropriate to asynchronously occurring events.

By using custom controls OCX, the control system of the present embodiment is a system that can be easily customized to match the peripheral devices selected by the user. It is also a control system that can execute the appropriate processes with high speed because two-way communication of properties, methods and events between custom controls OCX is assured.

The control system of this embodiment is described in further detail using a receipt printing function by way of example only.

One property of printer driver OCX 91, which is a level 3 OCX, is the status of receipt printer 112 (printer status). One method of printer driver OCX 91 outputs print commands to receipt printer 112. Printer driver OCX 91 also asynchronously generates events corresponding to real-time error status information supplied by receipt printer 112. This error status information includes no-paper or paper run-out and cover open status information.

Data from POS application program 71 is passed to receipt printer format conversion OCX 73, a level 1 object, and is converted thereby to the predetermined format before being passed to level 2 receipt printer control OCX 75. When receipt printer control OCX 75 calls the print execution method of printer driver OCX 91, printer driver OCX 91 sends the appropriate data and commands to receipt printer 112 and thereby prints a receipt. The receipt printer 112 executes the commands received from printer driver OCX 91 and returns printer status to printer driver OCX 91.

The printer driver OCX 91 interprets the returned status and terminates printing without generating an event if no error is indicated. It is also possible for the higher level receipt printer control OCX 75 to determine the result of the

printer process by referencing the printer status property. Communications for this referencing are handled by interface object 82a.

If an error during print command execution or while in the standby state, receipt printer 112 sends the error status to printer driver OCX 91. Printer driver OCX 91 generates an event in this case and notifies the higher level receipt printer control OCX 75 that an error has occurred through interface object 82a. Receipt printer control OCX 75 can execute specific processes for a given event and notify the format conversion OCX 73 and POS application program 71 through interface object 81a.

In addition to these properties, methods and events, other properties and events unique to slip printing are preferably provided in the driver OCX 92 for slip printer 113. For example, by adding a slip paper presence property to the driver OCX 92, higher level custom controls OCX and applications can regulate the timing at which the print command is issued. By adding paper insertion detection and paper end detection events to the event list, the application program can be controlled to smoothly execute the normal post-printing processes and error processing.

The cash drawer control OCX 77 communicates with cash drawer driver OCX 93. The cash drawer driver OCX 93 comprises a cash drawer open method, and a cash drawer status property that can be read to determine whether the cash drawer is open or closed. Events are issued when cash drawer errors occur and an open cash drawer is detected.

The customer display control OCX 78 communicates with display driver OCX 94, the methods of which include display commands specifying the display position and the display text string. Properties include the number of display columns, the display color, and other display specifications. Customer display control OCX 78 applies the display data according to these properties. Note that because the customer display is a dumb display, errors and similar events are rare. It is therefore possible to eliminate the event posting function from display driver OCX 94.

As described above, it is possible using the interface object according to the present invention to easily construct a control system comprising a multiple level hierarchical structure using plural custom controls OCX, and to respond rapidly to asynchronously occurring actions including errors. Note also that the control system of the present embodiment is an open control system that can be easily adapted to a variety of peripheral devices connected to a personal computer.

It is necessary to replace only the common driver object controlling the slip printer function with a new driver object corresponding to the new specifications when the slip printer in printer 111 is replaced with a slip printer having different printer function specifications (e.g., a different number of print columns or a different executable command set). Once the slip driver object has been updated, the peripheral device control system creates a custom control OCX from the common object for the new driver and automatically generates a control system suited to the new printer. This same principle applies when driver specifications change as a result of updating the printer driver, i.e., it is necessary to replace only the common object corresponding to the updated driver object, and it is not necessary to modify any other common objects or application program.

This is also true for the customer display and other peripheral devices, i.e., when the specifications change, the control system can be automatically modified by updating the common driver objects. When a POS system is constructed with bar code readers and other additional peripheral devices, the appropriate low-level custom controls OCX are created by the application program or high-level OCX if the common objects for controlling and driving these additional peripheral devices are provided somewhere in the personal computer system. As a result, a peripheral devices control system capable of driving the bar code reader and other peripheral devices is automatically configured.

It is also possible to construct a system using common application programs and to supply a common interface to all application programs irrespective of differences in personal computer hardware if the high-level OCX handling the interface to the application programs is capable of absorbing differences in specifications caused by the personal computer hardware. The control system is created by the application program with common objects in the personal computer, and a system appropriate to the personal computer and peripheral devices connected thereto is automatically created by the custom controls OCX constituting the control system.

The control system using the custom controls OCX of the present invention is thus an open system, and if the interface specifications are uniform and predetermined, application developers can develop and supply individual application programs without considering the specifications of the personal computer and peripheral devices connected thereto and without considering the method of connecting peripheral devices to the personal computer. Applications can therefore be developed in a short time and supplied at a low cost. This architecture also makes it possible for individual users to freely select the applications suited to their own objectives and environment irrespective of the personal computer or peripheral devices used.

Peripheral device suppliers can also supply general-purpose peripheral devices unrestricted to specific personal computer platforms or applications by providing with each device a low-level common object corresponding to the specifications of the supplied peripheral device. This also enables users to freely purchase peripheral devices suited to their own objectives and environment and easily construct a system designed for individual objectives.

It should be noted that there are many different types and numbers of peripheral devices that can be used, and while a POS system is described above as an example of a system in which various different types of peripheral devices are used according to individual user environments, the present invention is not limited to POS systems. Systems built

around personal computers increasingly combine peripheral devices of various manufacturers and specifications according to the objectives and capabilities of the user. Control systems using common control objects according to the present invention are systems that can be flexibly adapted to a variety of user intentions, and can be applied to a variety of systems other than POS systems.

Embodiment 2

Fig. 7 is a block diagram of the basic configuration of a control system according to the second embodiment of the present invention. In this embodiment IDispatch 13 is provided in a common object as the interface for receiving events. When events are communicated between custom controls OCX, the interface of the custom control OCX posting the event advises IConnectionPoint 12 about the event and IDispatch 13 receives the event for the OCX processing the event, i.e., the OCX posting the event passes an address identifier and opens a connection. Different identifiers can obviously be used under environments other than MFC, and different identifiers can be passed to either interface 12 or 13.

In addition to providing an event interface 13 in a custom control OCX to enable two-way communications between custom controls OCX, an interface object 16 connecting event interfaces is also derived from the COleDispatchDriver class in this embodiment. In other words, the control system of the present embodiment connects first OCX 10 and second OCX 11 by means of interface object 16 where first OCX 10 comprises IConnectionPoint 12 as the interface for posting events as described above and event IDispatch 13 as the interface for receiving events, and second OCX 11 downstream from first OCX 10 comprises IConnectionPoint 12 as the interface for posting events.

In addition to a function for supplying the properties and events of second OCX 11 to first OCX 10, interface object 16 of the present embodiment has a function for passing the address of event IDispatch 13 of first OCX 10 to IConnectionPoint 12 of second OCX 11. When first OCX 10 generates interface object 16 of the present embodiment, and interface object 16 generates second OCX 11 according to first OCX 10, a connection between IConnectionPoint 12 of second OCX 11 and event IDispatch 13 of first OCX 10 is opened and events posted by second OCX 11 can be received by first OCX 10. Two-way communication between second OCX 11 and first OCX 10 is therefore possible and there is no need for container application 20 to recognize second OCX 11. Second OCX 11 can also be supplied as an object unaffected by the container application 20.

Note that interface object 16 is derived from the COcxDispatchDriver class in the present embodiment. This COcxDispatchDriver class is an object class derived with multiple inheritance from the CCmdTarget class and the COleDispatchDriver class, and is described as follows.

```
class COcxDispatchDriver: public CCmdTarget, public COleDispatchDriver {...};
```

As described above, the CCmdTarget class is the class for deriving objects with a server function, and the COleDispatchDriver class is an object class that operates as either client or controller for the second OCX 11, and comprises a function for easily accessing methods and properties.

The COcxDispatchDriver class is an object class comprising the two main functions described above while inheriting and supporting all of the functions of both parent classes. The COcxDispatchDriver class is appropriate as an object class generating an interface object according to the present embodiment. Because there is no overlap between the operation and function of the CCmdTarget class and the COleDispatchDriver class, there is also no problem created by multiple inheritance.

The specification for the COcxDispatchDriver class is as follows:

```
Class COcxDispatchDriver: public CCmdTarget, public COleDispatchDriver {
COcxDispatchDriver::COcxDispatchDriver(); // create
public:                 // member data
IID m_IIDEvents;    // event interface ID
UINT m_nEvents;     // number of events
CDWordArray m_dispID; // array for converting dispatch ID using user-defined dispatch
                        map
public:                 // member functions
BOOL PrecreateDispatch(REFCLSID clsid,
    CStringArray& EntryNames,
    CDWordArray& DispIDs,
    CstringArray& ParamInfo,
    COleException *pError = NULL); // creates an object, and stores interface information.
BOOL EstablishConnection(LPUNKNOWN pUnkSink,
    LPCDMENTRY pDispMap,
    CStringArray& EntryNames,
    CDWordArray& DispIDs,
    CStringArray& ParamInfo,
     COleException *pError = NULL); // confirms event interface is same on posting and
                        receiving sides, connects the two sides.
void DestroyConnection(); // releases the connection, and deletes the object.
}
```

The interface object derived from the COcxDispatchDriver class in this embodiment has a function only for opening an event interface connection between custom controls OCX and, therefore, does not have an event processing function (event handler). The event handler is provided in the custom control OCX receiving the events. This is accomplished as follows.

A dispatch map is first described according to the external name of the event(s) returned by the OCX posting the events. The sequence of events in the dispatch map may be freely ordered and the correlation to the sequence of events actually received is established when the event connection is established. Note that it is necessary to fully describe all OCX events.

Dispatch map entries may be described using a macro call such as follows with the major parameters being, in sequence, the dispatch driver class name, external event name, event handler name, return values, and parameter data.

DISP_FUNCTION(_DSoprn, "ControlCompleteEvent", ControlCompleteEvent, VT_EMPTY, VTS_14 VTS_SCODE VTS_PBSTR VTS_I4)

The process for setting the connection between first OCX 10 and second OCX 11 by means of interface object 16 derived from the COcxDispatchDriver class is described below with reference to the flow charts in Fig. 2 and Fig. 3.

When interface object 16 is created and PrecreateDispatch is called, a second object OCX 11 defined by clsid is created and begins executing as shown in step 31 of Fig. 2. Subsequent operation is the same as in the first embodiment above and further description is omitted.

The process defining the correlation between the dispatch map and events is described next using the flow chart shown in Fig. 3.

This process is started by setting in pUnkSink the identifier (the address of the event IDispatch of first OCX 10 in this embodiment) of the interface to first OCX 10, which is the object receiving events, and calling EstablishConnection(). The counter fcnt is then cleared to zero in step 41. The counter fcnt is then read in step 42. Step 43 checks whether the event name EntryNames(fcnt) obtained from second OCX 11 is in the dispatch map pDispMap previously defined in the event handler of first OCX 10. If the event name EntryNames(fcnt) is in the dispatch map pDispMap, it is then confirmed in step 44 whether the parameter data ParamInfo(fcnt) matches the dispatch map pDispMap previously

defined in the event handler.

If step 44 returns YES, step 45 then defines array m_dispID for converting the dispatch ID DispIDs(fcnt) of the event obtained from second OCX 11 to an index in the dispatch map pDispMap previously defined in the event handler. The counter fcnt is then incremented in step 46, and the loop from step 42 to step 46 is repeated for each remaining event.

When the correlation to the dispatch map pDispMap is established, step 47 opens a connection for passing to the event handler of second OCX 11 the identifier, stored in pUnkSink, of the interface to first OCX 10. Because the present embodiment is described within the MFC environment, the address is passed to IConnectionPoint to establish the event connection. This makes it possible to pass events to first OCX 10 by calling the Invoke function of the second OCX 11. More specifically, it is possible for first OCX 10 to receive events passed from second OCX 11.

Step 48 checks whether the connection is successfully established. If so, the array m_dispID for converting the event dispatch ID is stored in step 49. If the preceding steps are all completed normally and the connection between interface object 16 and second OCX 11 is set, it is possible from step 50 to pass the methods and properties of second OCX 11 to first OCX 10, and to pass events from second OCX 11 to first OCX 10.

Fig. 4 is a flow chart of the process executed when the call Invoke() posting events of the IDispatch interface of interface object 16 is called from second OCX 11.

When Invoke() is called, the structure storing the event parameters and the dispatch ID defining the current event type are passed to first OCX 10. The event-receiving first OCX 10 then checks in step 60 whether the parameters are valid. If the parameters are valid, step 61 converts the dispIDMember array of events obtained from second OCX 11 according to the prepared event handler dispatch map pDispMap based on the conversion array m_dispID. The event handler corresponding to dispIDMember is then extracted from pDispMap in step 62.

Step 63 confirms whether the event handler is valid or not. If the event handler is valid, the event handler corresponding to the event is called in step 64 and the process defined by the event handler, e.g., a process informing the application program of the event, is executed.

Although the present embodiment has been described with only one second OCX 11 linked to first OCX 10, multiple second OCX 11 can be connected to first OCX 10. If there is only one event-receiving interface, the event array must contain identical events and all second OCXs 11 should preferably be the same type.

Application in a peripheral device control system

An example of a peripheral device control system applying the control system of the second embodiment is shown in Fig. 8. In the control system of this example, the custom controls OCX on at least the first and second levels are objects comprising an IDispatch interface as described above for communicating events. Therefore, by connecting high and low level objects using interface objects 83a, 83b, and 84a - 84d according to the present invention, high and low level objects can be linked using an interface capable of communicating events in addition to methods and properties. Further description of this application is omitted because the configuration and operation thereof are the same as in the peripheral device control system shown in Fig. 6 and described above.

Claims

1. A control system that is used in computers, said control system comprising:

   a first control object (10) and one or more second control objects (11; 11a, 11b), each of said first and second control objects having a respective first function for accessing one or more properties and for invoking methods calling functions implemented therein, and having a respective second function for posting events including asynchronously occurring actions, and
   one or more interface objects (15; 15a, 15b) each having a respective third function for communicating properties and/or methods between said first control object (10) and a respective second control object (11; 11a, 11b), and having a respective fourth function for posting events from said respective second control object to said first control object.

2. A control system according to Claim 1 wherein said first control object (10) is associated with a plurality of said second control objects (11a, 11b), and a respective interface object (15a, 15b) is uniquely associated with a respective second control object (11a, 11b), each interface object (15a, 15b) comprising an independent identification means that can be referenced by said first control object (10).

3. A control system according to Claim 1 or 2 wherein a respective second control object (11; 11a, 11b) posts plural events according to a first array and a respective interface object (15; 15a, 15b) comprises conversion means for converting said plural events to a second array.

4. A control system according to Claim 3 wherein each interface object (15; 15a, 15b) is associated with a respective second control object (11; 11a, 11b) and each interface object comprises plural event processing means invoked according to said second array.

5. A method for providing a control system according to any one of Claims 1 through 4, comprising the steps of:

said first control object (10) creating said one or more second control objects (11; 11a, 11b), and
said first control object (10) creating for each of said second control objects (11; 11a, 11b) a respective interface object (15; 15a, 15b).

6. A method according to Claim 5 further comprising the steps of:

said first control (10) object accessing an independent identification means for each interface object (15; 15a, 15b), and
said second control object (11; 11a, 11b) posting said identification means with said events to said first control object (10).

7. A method according to Claim 5 further comprising the steps of:

creating a third array when a respective interface object (15; 15a, 15b) is created,
posting a plurality of events in a respective second control object (11; 11a, 11b) according to a first array,
invoking, in response to each of said plurality of events, a corresponding one of a plurality of event processing means in said respective interface object (15; 15a, 15b) according to a second array, and
converting said first array to said second array according to contents of said third array.

8. A control system that is used in computers, comprising:

a plurality of control objects (10, 11) each having a respective first function for accessing one or more properties and for invoking methods calling functions implemented therein, and a first interface (12) for posting events including asynchronously occurring actions, a second one (11) of said control objects being created and controlled by a first one (10) of said control objects, wherein

said first control object (10) comprises a second interface (13) for receiving events, and
said control system further comprises an interface object (16) having
a function for communicating properties and/or methods between said first control object (10) and said second control object (11), and
a function for passing an identifier of said second interface (13) of the first control object (10) to the first interface (12) of the second control object (11) or for passing an identifier of the first interface (12) of the second control object (11) to the second interface (13) of the first control object (10).

9. A control system according to Claim 8 wherein the second control object (11) posts plural events according to a first array, and
the first control object (10) comprises conversion means for converting said plural received events to a second array.

10. A method of providing a control system according to claim 8 or 9 comprising the steps of:

said first control object (10) creating said interface object (16),
said interface object (16) creating said second control object (11), and
establishing a connection between the second interface (13) of the first control object (10) and the first interface (12) of the second control object (11).

11. A method according to Claim 10 for providing a control system according to claim 9, wherein said step for establishing a connection further creates a third array for converting the second array to the first array.

12. A peripheral device control system (70) for controlling peripheral devices (112, 113, 115) of an application system by means of a control system according to any one of claims 1 to 4, 8 or 9, wherein each of a plurality of control object groups includes a first control object (75-78), a second control object (91-94)) and an interface object (82a-82d; 84a-84d), and wherein each second control object (91-94) is associated with a corresponding one of said

peripheral devices (112, 113, 115).

13. A system (70) according to Claim 12 wherein the properties of a respective second control object (91-94) includes attribute values corresponding to unique specifications of the associated peripheral device (112, 113, 115),

the methods of the respective second control object (91-94) include commands specific to the associated peripheral device (112, 113, 115), and

the events of the second control object (19-94) correspond to actions occurring asynchronously in the associated peripheral device (112, 113, 115).

14. A control system (70) according to Claim 12 or 13 wherein the first control object (75-78) converts the properties, methods and/or events of the second control object (91-94) communicated through said interface object (82a-82d; 84a-84d) to a universal specification.

15. A system (70) according to Claim 12, 13 or 14 wherein the application system is a point-of-sale control system.

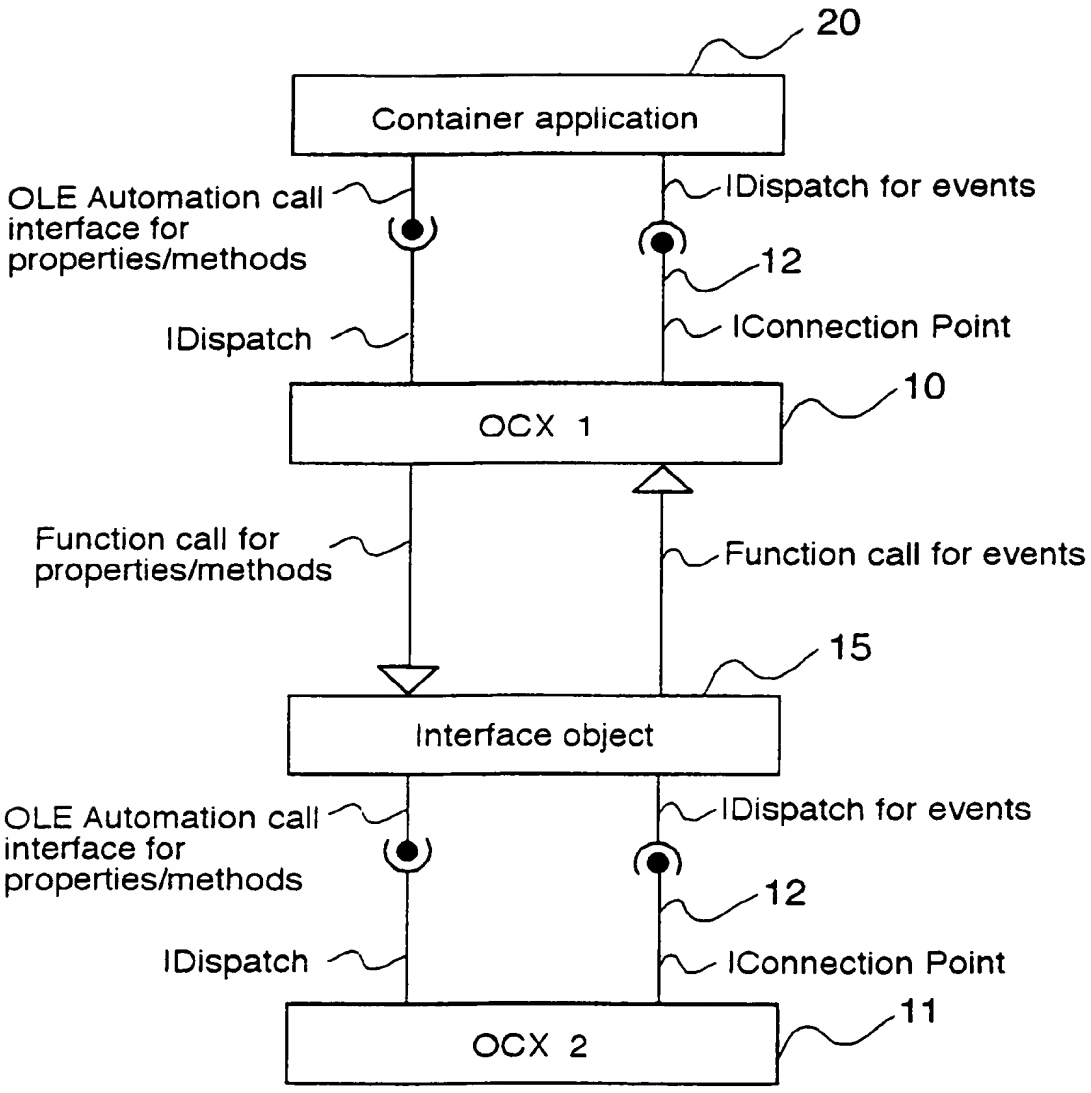

FIG. 1

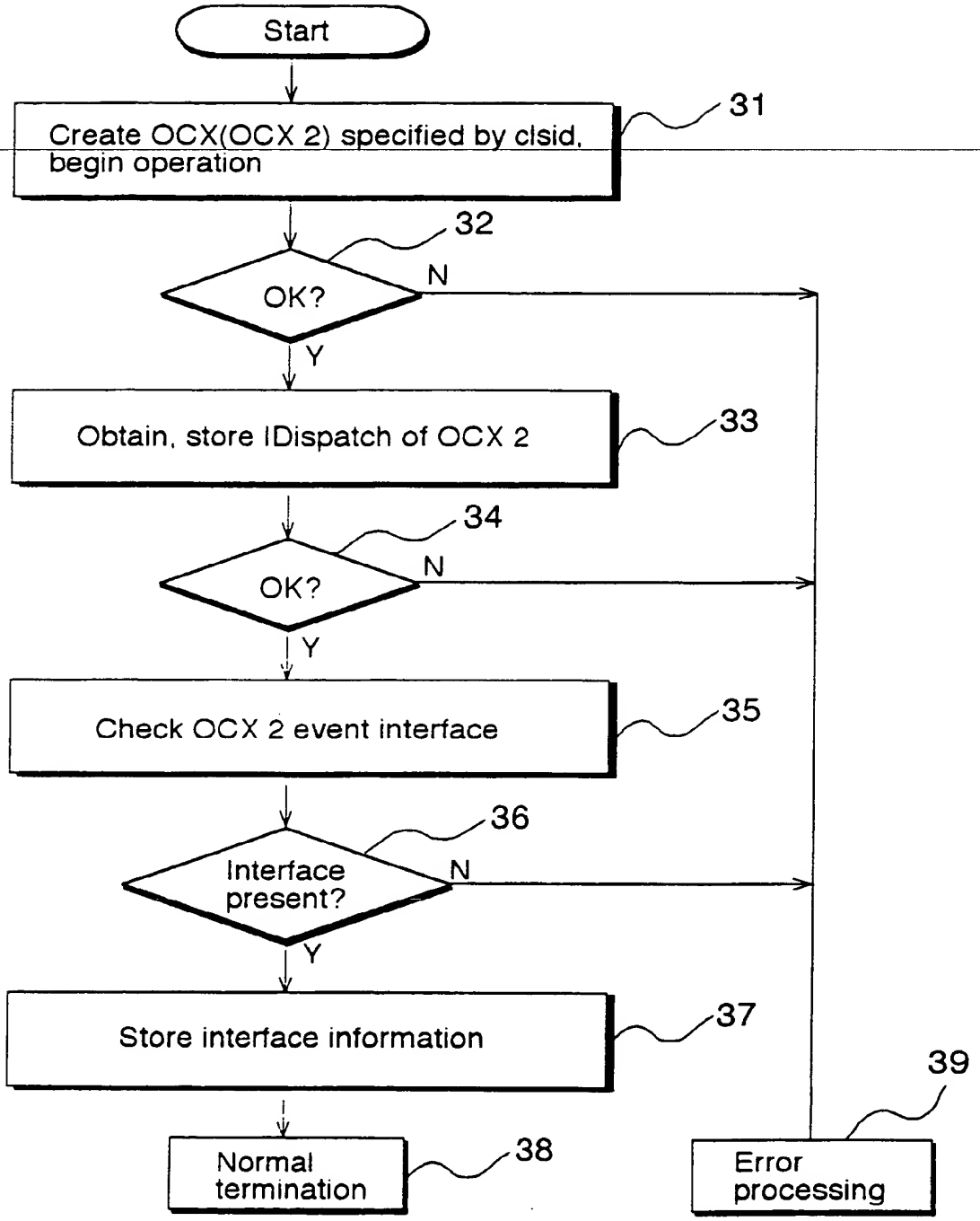

FIG. 2

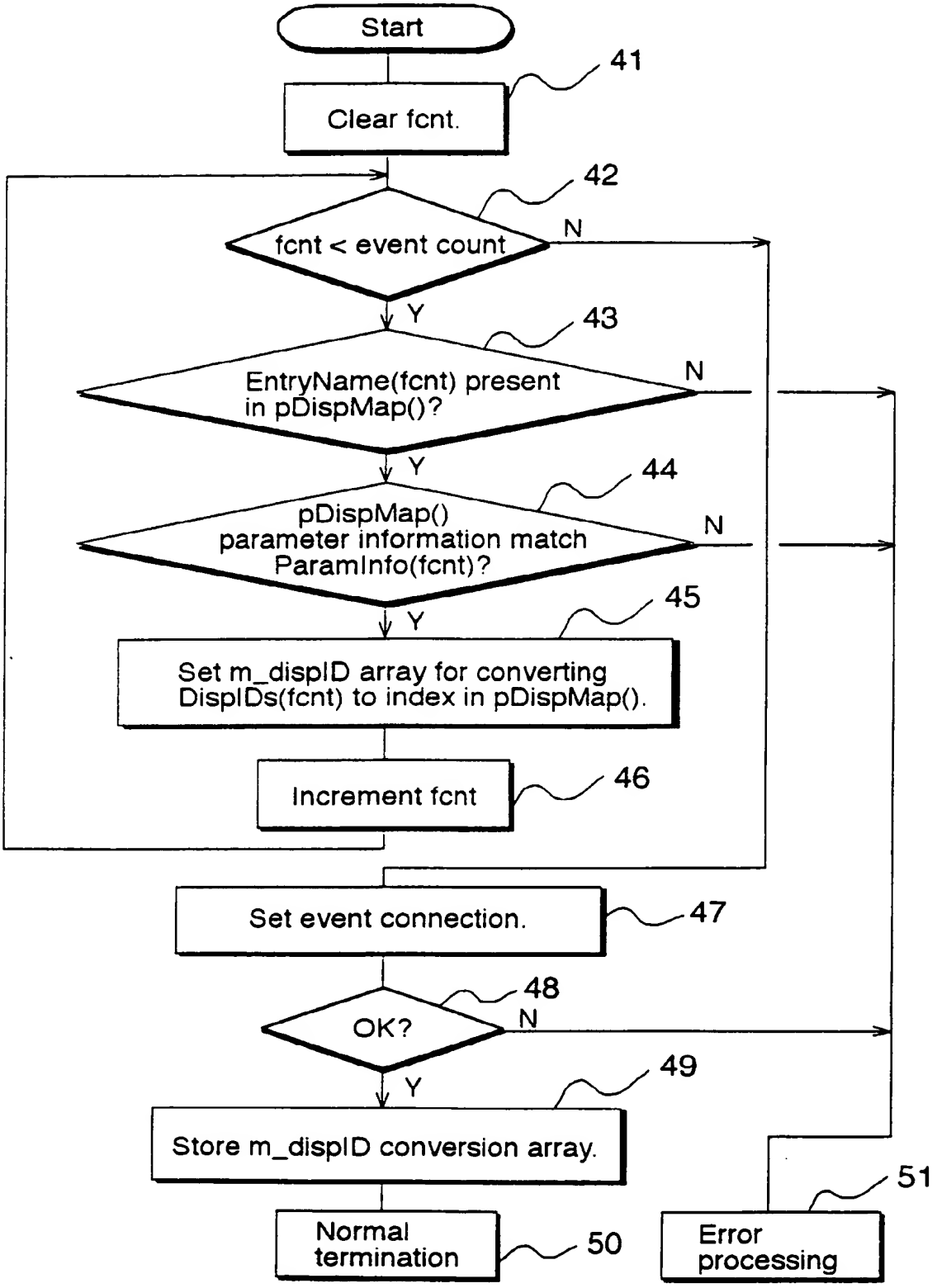

FIG. 3

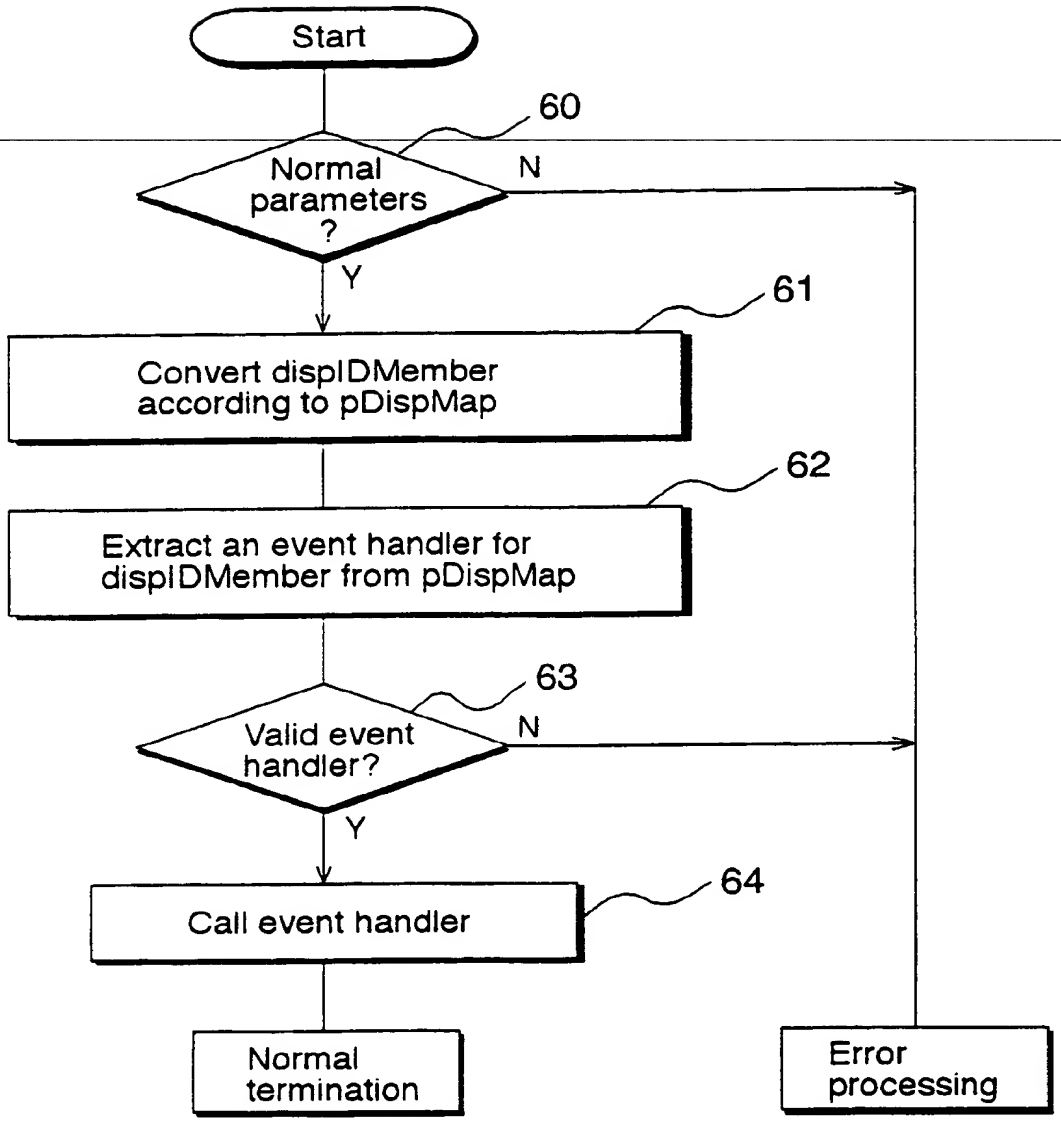

FIG. 4

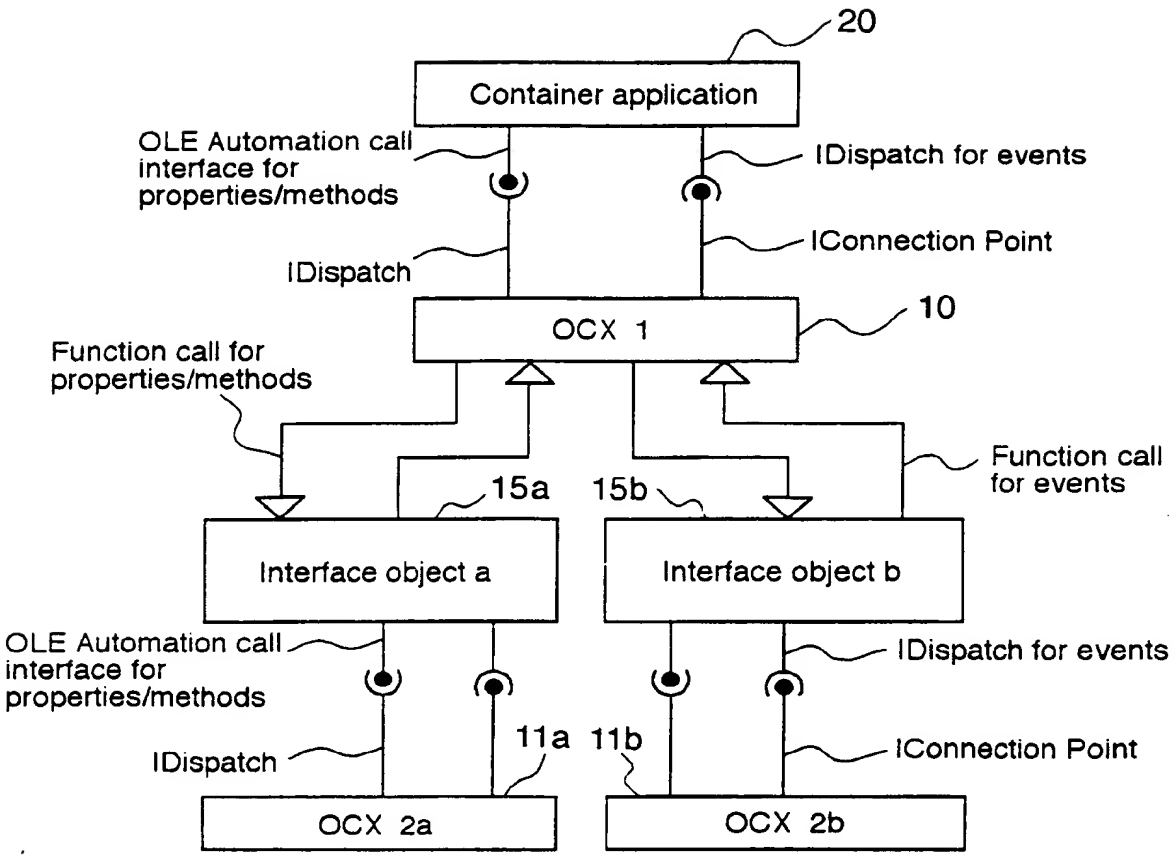

FIG. 5

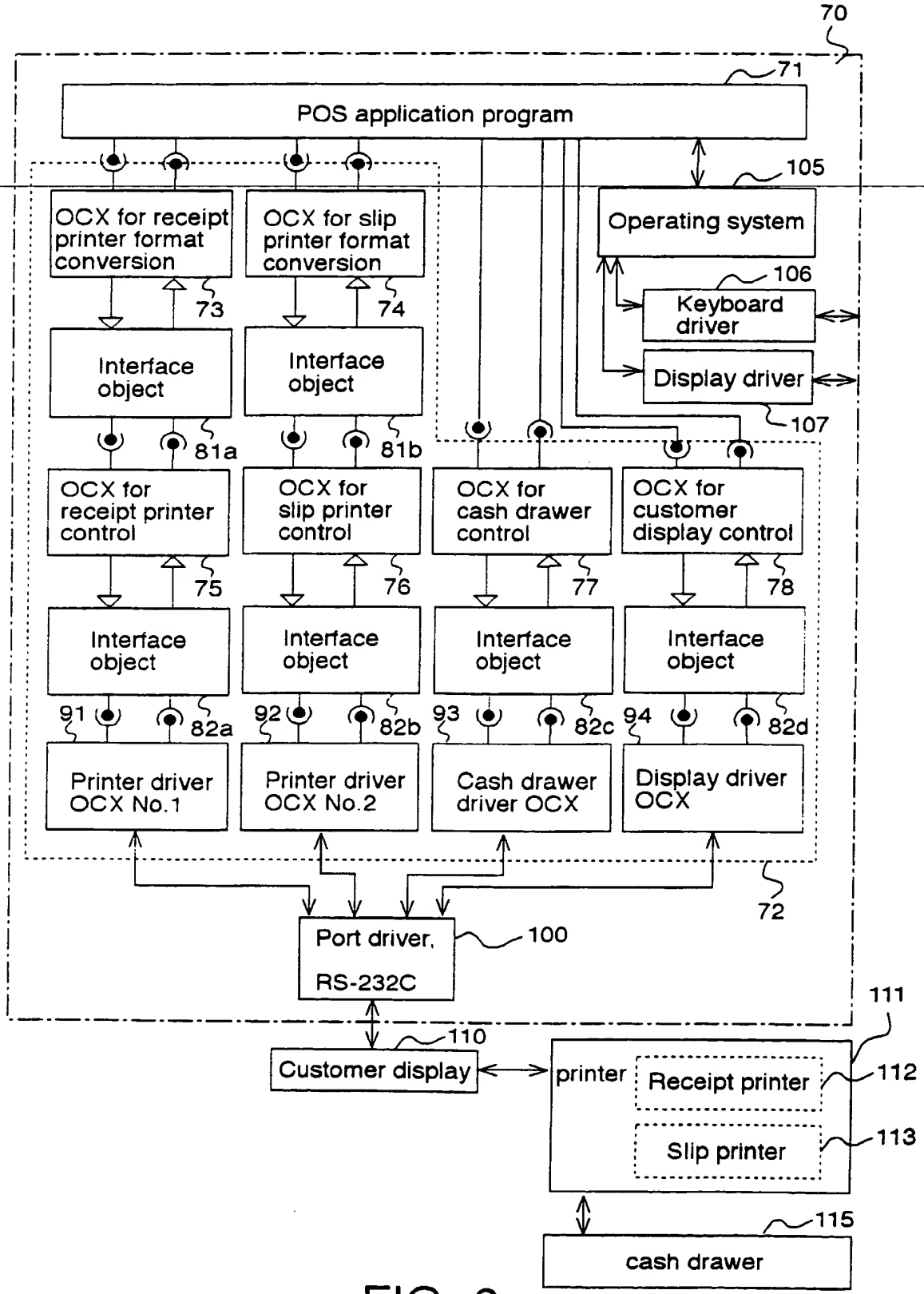

FIG. 6

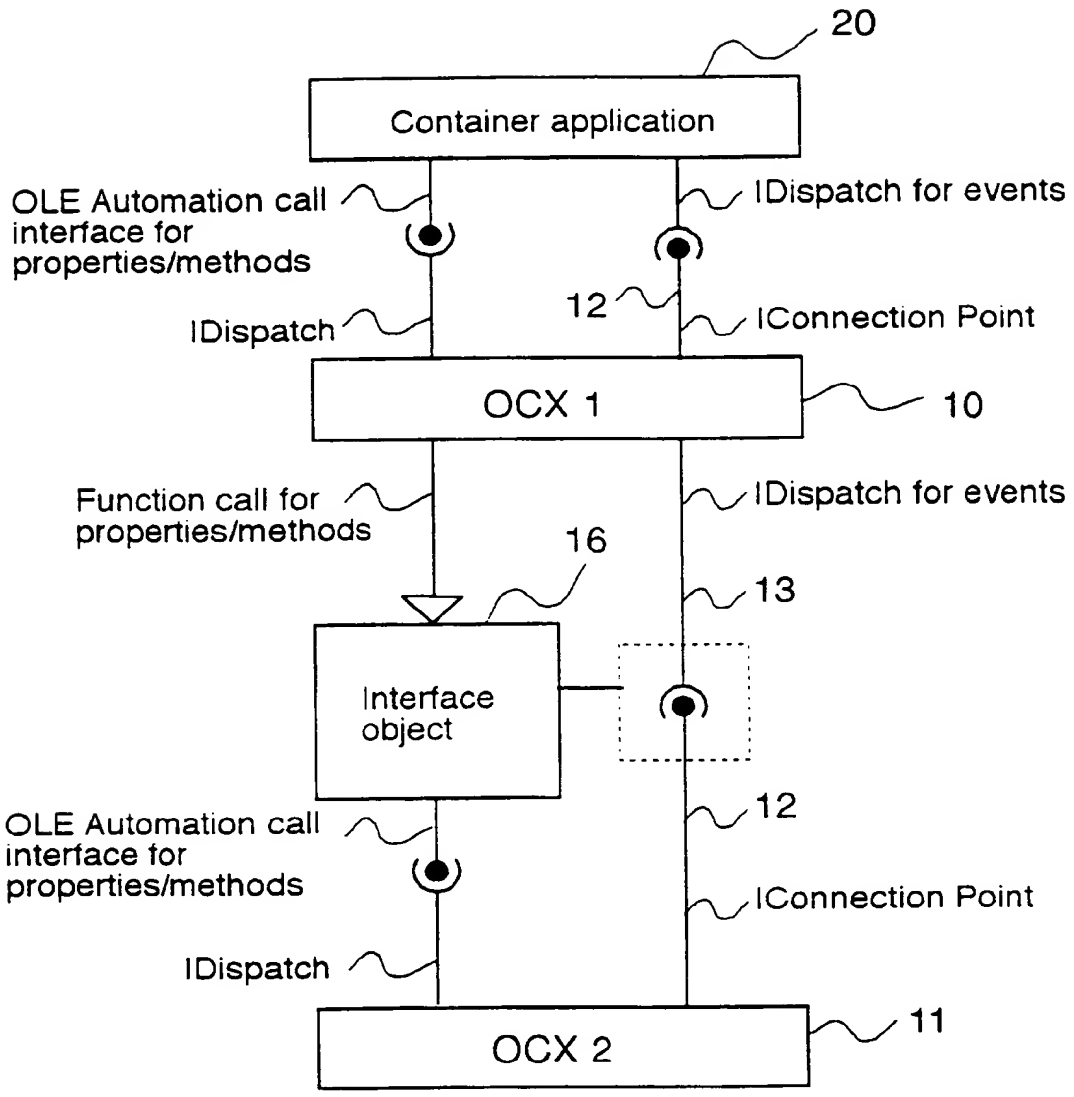

FIG. 7

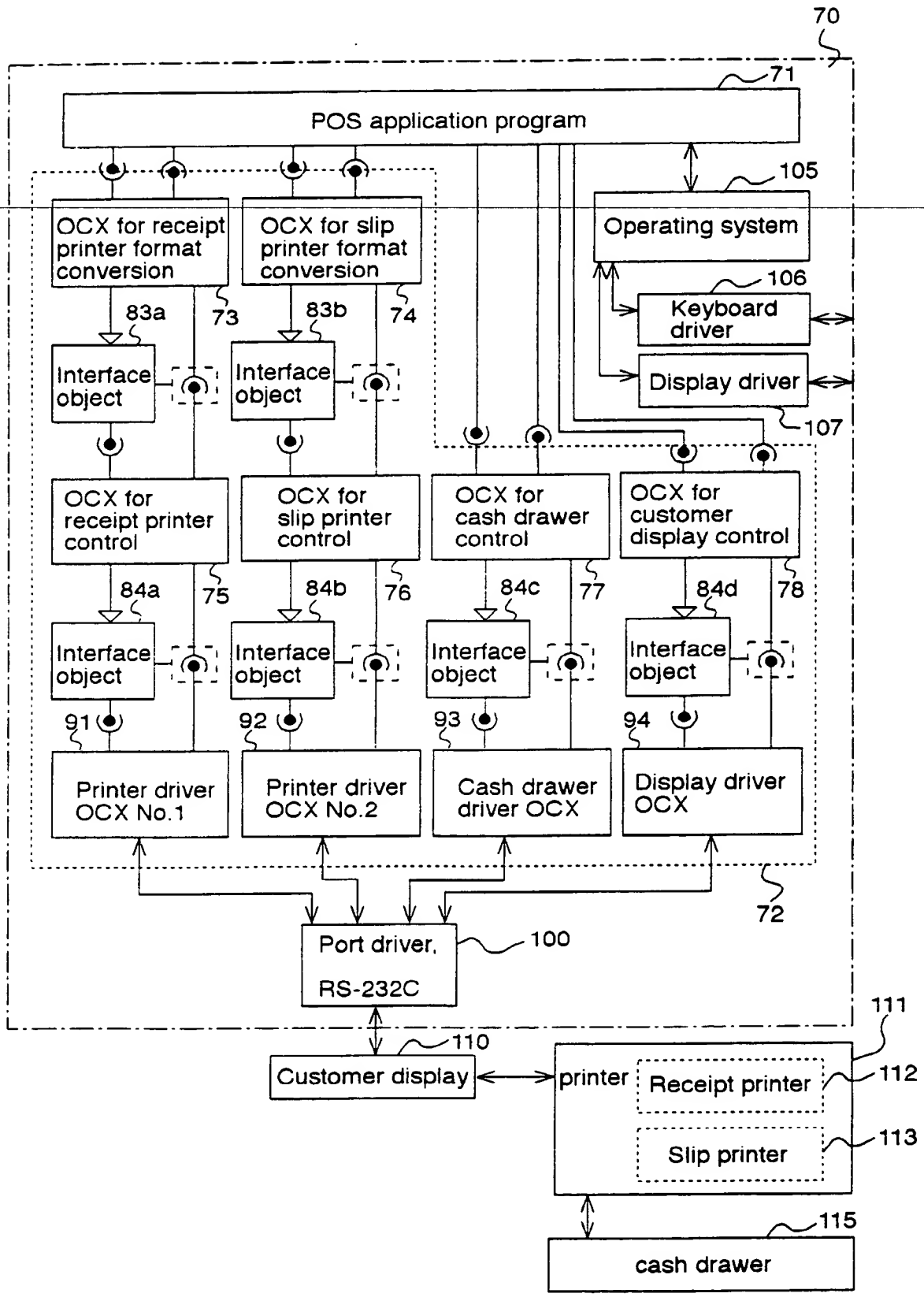

FIG. 8

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 11 4176

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	EP-A-0 660 231 (MICROSOFT) 28 June 1995 * the whole document * ---	1-14	G06F9/44 G06F9/46
A	DR DOBBS SPECIAL REPORT, vol. 19, no. 16, CA,USA, pages 42-49, XP002021208 KRAIG BROCKSCHMIDT: "OLE Integration Technologies" Winter 1994/1995 * page 46, middle column, line 14 - page 47, middle column, line 9; figure 6 * * page 49, left-hand column, last paragraph - right-hand column, line 31 * -----	1-15	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	16 December 1996	Fonderson, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)